Hardware Implementation of High-Performance Polynomial Multiplication for KEM Saber

Yazheng Tu[1], Pengzhou He[1], Chiou-Yng Lee[2], Danai Chasaki[1], and Jiafeng Xie (corresponding author)[1]
[1]: Department of Electrical and Computer Engineering, Villanova University, Villanova, PA, USA 19085.
[2]: Department of Computer Information & Network Engineering, Lunghwa University of Science and Technology, Taiwan
[1]Email: ytu1,phe,danai.chasaki,jiafeng.xie@villanova.edu. [2]Email: pp010@gm.lhu.edu.tw.

2022 IEEE International Symposium on Circuits and Systems (ISCAS) | 978-1-6654-8485-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISCAS48785.2022.9937606

***Abstract*—Recent advances in quantum computing have initiated a new round of cryptosystem innovation as the existing public-key cryptosystems are proven to be vulnerable to quantum attacks. Several types of cryptographic algorithms have been proposed for possible post-quantum cryptography (PQC) candidates and the lattice-based key encapsulation mechanism (KEM) Saber is one of the most promising algorithms. Noticing that the polynomial multiplication over ring is the key arithmetic operation of KEM Saber, in this paper, we propose a novel strategy for efficient implementation of polynomial multiplication on the hardware platform. First of all, we present the proposed mathematical derivation process for polynomial multiplication. Then, the proposed hardware structure is provided. Finally, field-programmable gate array (FPGA) based implementation results are obtained, and it is shown that the proposed design has better performance than the existing ones. The proposed polynomial multiplication can be further deployed to construct efficient hardware cryptoprocessors for KEM Saber.**

I. Introduction

It is proven that the current public-key cryptosystems are vulnerable to the attacks launched from powerful quantum computers executing Shor's algorithm [1-3] (in the next 15 to 20 years). Post-quantum cryptography (PQC) has thus gained substantial attention from various communities. The National Institute of Standards and Technology (NIST) has already started the PQC standardization process [3]. The lattice-based key encapsulation mechanism (KEM) Saber is among one of the recently released third-round PQC finalists [4].

KEM Saber is built on the module learning-with-rounding (MLWR) problem [5-6], which is a module variant of the LWR problem [7]. LWR is a variant of the learning with errors (LWE) problem [8-9], where the errors are produced by a rounding operation rather than from a random distribution [5]. Since its initial introduction in [5], important works have been released on KEM Saber about its quantum and classic security levels and related computational complexity. Along with the NIST third-round PQC standardization process, research on KEM Saber has gradually switched to implementations, especially the hardware implementation strategies [10-16].

Existing Works. There are two types of hardware design strategies for KEM Saber: the hardware-software co-design and the full hardware design. The former type can be seen in a recent paper of [17], where the authors proposed to use the Toom-Cook approach for efficient implementation of the polynomial multiplication of KEM Saber. Another hardware-software co-design is proposed in [18], where the design has better performance than the previous one but with larger area-complexity. A very recent report suggested using the number theoretic transform (NTT) approach for the implementation of polynomial multiplication within Saber on a RISC-V accelerator [19]. For the full hardware design type, the first design is proposed in [20]. Then, a new Karatsuba algorithm-based Saber cryptoprocessor is reported in [21]. After that, a new optimized polynomial multiplier for Saber is presented in [22]. Efficient implementations of KEM Saber can also be seen in the very recent works of [23] and [24], respectively. Overall, these works represent the major efforts in the field.

It is noted that the polynomial multiplication over ring is the major arithmetic operation of KEM Saber. However, efficient implementations of high-performance polynomial multiplication (especially hardware designs) are very limited: (i) The proposed existing works such as [20], [22] still use the traditional schoolbook algorithm and no other algorithmic derivations have been made; (ii) Not many specific designs for high-performance polynomial multiplications are reported though this type of implementation is critical to the overall efficiency of the final cryptoprocessor; (iii) The existing designs mostly use the sign magnitude format to process the data, which actually requires extra resource for data type transferring. Based on this consideration, in this paper, we propose a novel hardware implementation of polynomial multiplication for KEM Saber with high-performance. Key contributions are:

- We have presented the proposed algorithmic operation for the polynomial multiplication of KEM Saber.
- We have introduced the corresponding hardware architecture with thorough internal structural descriptions.
- We have given the final comparison to demonstrate the superior performance of the proposed design.

The rest of the paper is organized as follows. Section II focuses on the preliminary knowledge. Section III presents the proposed algorithmic derivation. Section IV introduces the proposed structure. The comparison is provided in Section V, and the conclusion is given in Section VI.

II. Preliminary Knowledge

KEM Saber: MLWR-based Scheme. LWR is a variant of the LWE problem, where the errors are generated by a rounding operation [4], i.e., the samples are produced through $(a, b = \lfloor \frac{p}{q}\langle a, s\rangle \rceil_p) \in \mathbb{Z}_q^n \times \mathbb{Z}_p$. The MLWR-based scheme is based on the module LWR problem.

Saber is an IND-CCA secure KEM built on the hardness of the MLWR problem, which achieves both classical and

quantum security [4]. Firstly, Saber is introduced as a Chosen Plaintext Attack (CPA) secure public-key scheme. Then, a Chosen Ciphertext Attack (CCA) secure KEM Saber is constructed through the Fujisaki-Okamoto transformation [4], [25].

Saber public-key encryption scheme involves three operational phases, namely key generation, encryption, and decryption. During the key generation phase, a public matrix of polynomials A and a secret vector of polynomials s are produced to generate the vector b through scaling and rounding operations (public key contains A and b, and secret key is s). During the encryption phase, the ciphertext is generated through the operation of $V = s'b$. During the decryption phase, the original message is obtained through the approximation of V (from sb'). KEM Saber is built based on the encryption scheme, and the detailed information can be seen at [4], [5]. Note that polynomial multiplication determines the overall performance (or complexity) of KEM Saber.

Security Ranks. There are three security ranks of parameters settings, i,e., LightSaber, Saber, and FireSaber. The polynomial degree is $N = 256$ and moduli $q = 2^{13}$ and $p = 2^{10}$. KEM Saber uses secrets sampled from the binomial distribution: [-5,5] (LightSaber), [-4,4] (Saber), and [-3,3] (FireSaber) [5].

Polynomial Multiplication for KEM Saber. Polynomial multiplication is the key operation of KEM Saber, where one polynomial contains small-size coefficients ([-5,5]) and the other polynomial has coefficients of either 10-bit or 13-bit.

III. The Proposed Algorithmic Operation for Polynomial Multiplication of KEM Saber

Without loss of generality, we can define that one polynomial consists of coefficients of 4-bit (generated by the binomial sampling) and the other polynomial has coefficients of 13-bit (the polynomial with 10-bit coefficients is also covered here).

Define the polynomial multiplication for KEM Saber as

$$W = GD \bmod f(x), \tag{1}$$

where $f(x) = x^N + 1$, $W = \sum_{i=0}^{N-1} w_i x^i$, $G = \sum_{i=0}^{N-1} g_i x^i$, and $D = \sum_{i=0}^{N-1} d_i x^i$ (g_i, d_i, and w_i are 4-bit, 13-bit, and 13-bit integers over ring $\mathbb{Z}_q/(x^N+1)$, respectively).

Generally, we can have

$$W = \sum_{i=0}^{N-1} G d_i x^i \bmod f(x) = \sum_{i=0}^{N-1} (Gx^i \bmod f(x)) d_i, \tag{2}$$

which turns to be the accumulation of $(Gx^i \bmod f(x))d_i$ (N cycles). This strategy has also been used in the existing reports such as [20], [22]. Though the existing designs have used the sign magnitude format for computation to ease the implementation process (especially the modulo operation involved sign inversion and multiplications), extra resources are required to transfer the data in/out of polynomial multiplication into two's complement form for further processing [22].

Proposed Mathematical Derivation Strategy. Based on this consideration, we propose to: (i) design the polynomial multiplication with full two's complement representation (eliminate extra resource usage); (ii) move the modulo operation outside of the point-wise multiplications (reduce propagation delay).

Let us define $N = hk$ (h and k are integers), we can have

$$D = \sum_{j=0}^{h-1} D_j x^{kj}, \tag{3}$$

where $D_j = d_{kj} + \cdots + d_{kj+k-1} x^{kj+k-1}$. Then, we have

$$W = \sum_{j=0}^{h-1} G D_j x^{kj} \bmod f(x) = \sum_{j=0}^{h-1} G^{(kj)} D_j \bmod f(x), \tag{4}$$

where we defined $Gx^{kj} \bmod f(x) = G^{(kj)}$. The final output becomes the accumulation of $G^{(kj)}D_j$ (followed by a modulo operation). We thus have the proposed algorithmic process as

Algorithm 1: Proposed polynomial multiplication algorithmic operation for KEM Saber

Inputs : G and D are integer polynomials. // the actual bit-width of the coefficients follow (1).
Output: $W = GD \bmod (x^N + 1)$.

Initialization step
Process the inputs G and D, i.e., $D = \sum_{j=0}^{h-1} D_j x^{kj}$ and $G^{(0)} = G$;
$\overline{W} = 0$;
Main step
for $j = 0$ *to* $h-1$ **do**
  $\overline{W} = \overline{W} + G^{(kj)} D_j \bmod f(x)$; // follow (4)
  Obtain $G^{(k(j+1))}$ from $G^{(kj)}$ (follow the operation of $G^{(kj)} = Gx^{kj} \bmod f(x)$);
end
$W = \overline{W}$;
Final step
Deliver all the coefficients of output W;

Simple Example. Assume $N = 4$ and $k = 2$, we can have

$$\begin{aligned} G^{(0)} &= G = g_0 + g_1 x + g_2 x^2 + g_3 x^3, \\ G^{(1)} &= Gx^2 \bmod f(x) \\ &= (g_0 x^2 + g_1 x^3 + g_2 x^4 + g_3 x^5) \bmod f(x) \\ &= -g_2 - g_3 x + g_0 x^2 + g_1 x^3, \end{aligned} \tag{5}$$

where $f(x) = x^4 + 1 \Rightarrow x^4 \equiv -1$ has been substituted. The final output becomes $W = [G^{(0)}D_0 + G^{(1)}D_1] \bmod f(x)$ and

$$\begin{aligned} &G^{(0)}(d_0 + d_1 x) \bmod f(x) = g_0 d_0 + (g_1 d_0 + g_0 d_1)x + \\ &(g_2 d_0 + g_1 d_1)x^2 + (g_3 d_0 + g_2 d_0)x^3 + g_3 d_1 x^4 \bmod f(x) \\ &= (g_0 d_0 - g_3 d_1) + (g_1 d_0 + g_0 d_1)x + (g_2 d_0 + g_1 d_1)x^2 \\ &+ (g_3 d_0 + g_2 d_0)x^3, \end{aligned} \tag{6}$$

where $G^{(1)}(d_2 + d_3 x) \bmod f(x)$ can be similarly calculated. Overall, $G^{(0)}(d_0 + d_1 x)$ (followed by the modulo operation) is calculated first and then accumulated with $G^{(1)}(d_2 + d_3 x)$ (along with a following modulo operation) to obtain W.

IV. Proposed Architecture

The proposed polynomial multiplication architecture for KEM Saber is shown in Fig. 1, where the architecture consists of three main components, namely the circular shift-register

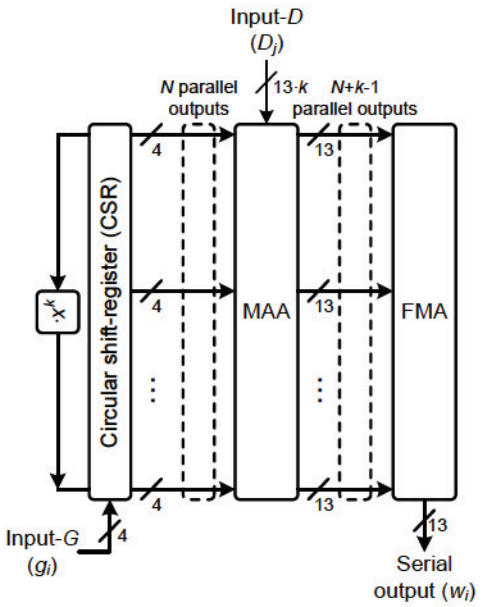

Fig. 1: The proposed polynomial multiplication (KEM Saber).

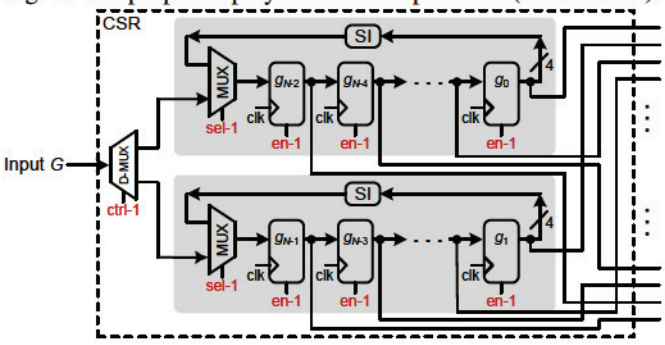

Fig. 2: CSR ($k = 2$, values in the registers are initially loaded).

(CSR), the multiplication and addition (MAA) component, and the final modular and accumulation (FMA) component.

CSR component. According to Step 5 of Algorithm 1, we have to execute the operation of obtaining $G^{(kj+k)}$ from $G^{(kj)}$ every clock cycle. Since $G^{(kj)} = Gx^{kj} \bmod f(x)$, we can have $G^{(kj+k)} = Gx^{kj+k} \bmod f(x) = Gx^{kj}x^k \bmod f(x) = G^{(kj)}x^k \bmod f(x)$, i.e., obtaining $G^{(kj+k)}$ from $G^{(kj)}$ requires the operation of $\cdot x^k \bmod f(x)$. To execute this operation, a novel CSR is used for the proposed architecture as shown in Fig. 2 (for simplicity of discussion, we have presented the case of $k = 2$, which can be easily extended to other values of k). For this example, we have used one DE-MUX (D-MUX), two MUXes, N 4-bit registers, and two sign inverters (SIs). In the loading stage, the even-order (and the odd-order) coefficients are respectively loaded into the corresponding registers through the functions of D-MUX and MUXes. After that, two MUXes operate to let the values loaded in the registers shift in a circular format (through a sign inverter (SI), as shown in Fig. 3). The SI works to invert the values in the right-left registers under two's complement format. All the outputs are finally combined to form N parallel outputs to be fed to the following MAA component.

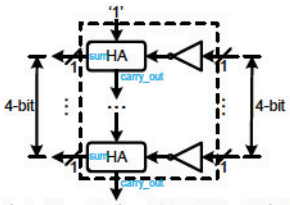

Fig. 3: The details of the SI (HA: 1-bit half adder).

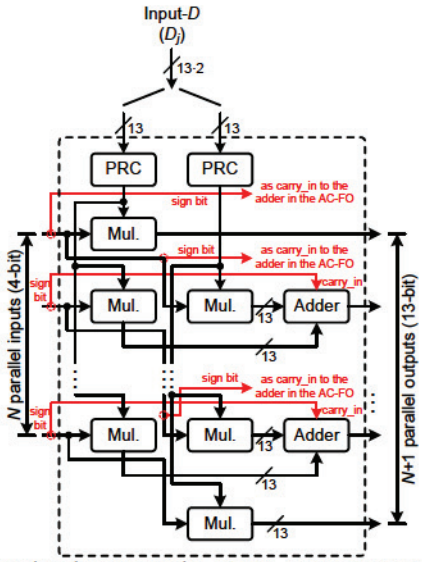

Fig. 4: The details of the MAA component ($k = 2$).

MAA component. The MAA component executes the operation of $G^{(kj)}D_j$ (connecting Step 4 of Algorithm 1 and also the provided example). We have proposed a new MUX-based multiplier to save the resource usage under two's complement representation format, which is different from that of [22]. We have again used $k = 2$ to present the detailed structure, as shown in Fig. 4. Two inputs from D (13-bit) are multiplied with N parallel outputs from the CSR component, respectively. To save the resource usage, the related multiplicand coefficients are pre-calculated in the pre-computing (PRC) cell. Since our polynomial multiplier targets all three security ranks of KEM Saber, we have used an 11-to-1 MUX in the multiplication (Mul.) cell, as shown in Fig. 5. Note that the sign bit from 4-bit parallel inputs are used as carry-in for the following adders (including the FMA component, as shown in Fig. 6) to meet the two's complement representation requirement. Besides that, due to the order difference of the two inputs from D (see the example of (5)), we have used $(N-1)$ adders to produce $(N+1)$ parallel outputs (13-bit). This structure can be easily extended to other values of k.

FMA component. The internal structure for the FMA component is shown in Fig. 6 ($k = 2$), where it consists of a final modulo and an accumulation & output sub-components. The final modulo sub-component is relatively simple, i.e., the $(N+1)$th output from the MAA component is inverted and adds with the first output (carry-in is set as '1' according to the two's complement format) while the rest $(N-1)$ parallel inputs remain the same. The accumulation & output sub-component involves N parallel accumulation-final output (AC-

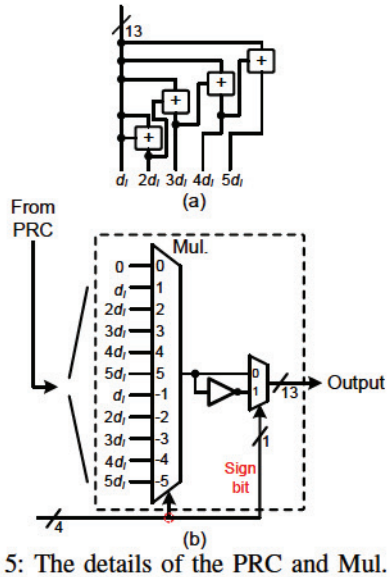

Fig. 5: The details of the PRC and Mul. cells.

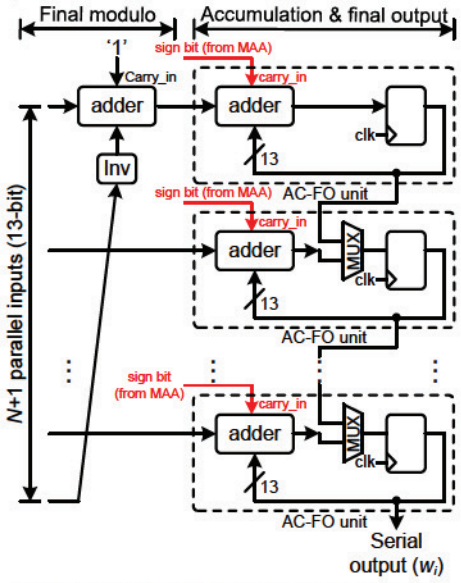

Fig. 6: The details of the FMA component ($k = 2$).

FO) units, where each unit contains an adder, a register, and a MUX. During the accumulation process, the MUX is set to channel that the output of the register is added with the newly arrived input from the final modulo sub-component and then stored in the register again. After the accumulation process is finished (N cycles), the MUX sets to another channel that all the values stored in the registers can be serially delivered out.

V. Implementation and Comparison

The proposed structure of Fig. 1 needs N cycles of input loading, N/k cycles of computation, and N cycles of final output delivering. We have coded the proposed polynomial multiplication and obtained the implementation results.

Experimental setup. We have set the experiment as follows: (i) We have coded the proposed architecture with VHDL and implemented the design on the Xilinx UltraScale+ XCZU9EG-FFVB1156-2 FPGA through Vivado 2019.2; (ii) We have used $N = 256$ and have verified the correctness of the code through ModelSim; (iii) We have selected $k = 2$ and $k = 4$ for the proposed design to obtain the implementation performance. The obtained results, including the number of LUT, FF, maximum frequency (MHz), and power consumption (dynamic power, W), are listed in Table I.

TABLE I: Comparison of the Implementation Results

design	LUT	FF	Fmax	delay[1]	ADP*	power[3]
UltraScale+ XCZU9EG-FFVB1156-2 FPGA device						
Ext. [20][2]	29,141	4,907	250	512	14,920	-
Ext. [22]	22,118	4,920	250	512	11,324	-
Pro. ($k = 2$)	23,784	4,404	270	474	11,274	1.116
Pro. ($k = 4$)	44,176	4,420	256	250	11,309	2.556

The designs in [20], [22] use sign magnitude representation to ease the implementation, but require extra resources for data type transferring (into two's complement form) between the polynomial multiplication and other components within KEM Saber (this resource usage is not reported here). For a fair comparison, we only list the existing designs with a latency of 128 (the same as proposed design of $k = 2$).
Unit for delay: ns. *: ADP=#LUT×delay ($\times 10^3$).
[1]: delay is calculated as latency×(1/Fmax), where the latency refers to the computation time (input loading and output delivering are not included).
[2]: this implementation results come from [22], where the same authors have re-implemented their design in [20].
[3]: The existing designs do not report the power consumption.

The LUT utilization of the proposed design increases almost proportionally when k increases from 2 to 4 while the number of FF remains nearly the same. This is because the registers used in the proposed structure, mainly from the CSR and the AC-FO units, are very stable even when k changes. Meanwhile, the increase of k has impacts on the Fmax.

We have also listed the major works in the field, i.e., the polynomial multiplications of [20], [22] for comparison, as shown in Table I. It is shown that the proposed design ($k = 2$) has the least area-delay product (ADP) among all the reported designs, e.g., 24.4% smaller than the existing design [20]. Besides that, one has to note that the proposed design covers all security ranks of KEM Saber while the existing ones of [20], [22] only cover two. The efficiency of the proposed design comes from two aspects: (i) the MAA component has used a new data processing method, which reduces involved propagation delay; (ii) the multipliers used in the MAA component have used a new resource sharing scheme, which lowers the overall area occupation.

Finally, we want to emphasize that the proposed design is wholly built on the two's complement representation system, while the existing designs ([20], [22]) are based on the sign magnitude format (require extra resource usage for data type transferring [20]). Future works may focus on algorithmic and architectural innovations and side-channel attacks [26-29].

VI. Conclusion

This paper presents a novel implementation of polynomial multiplication for KEM Saber on FPGA platform. We have firstly presented the proposed algorithmic process for polynomial multiplication. Then, the details of the structure are provided. Implementation and comparison are finally given to confirm the efficiency of the proposed design.

VII. Acknowledgement

The work of J. Xie is supported by NSF SaTC-2020625 and NIST-60NANB20D203.

REFERENCES

[1] D. Micciancio. Lattice-based cryptography. *Encyclopedia of Cryptography & Security*, 2011.

[2] W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. *Symp. Founda. of Computer Science*, pp. 124-134, 1994.

[3] Post-quantum cryptography round 3 submissions. https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions

[4] J.-P. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren, "Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM," vol. 10831, pp. 282-305, 2018.

[5] G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, J. Kelsey, Y.-K. Liu et al., "Status report on the second round of the NIST post-quantum cryptography standardization process," US Department of Commerce, NIST (2020).

[6] J.-P. D'Anvers et al., "SABER. Proposal to NIST PQC Standardization,"Round2, 2019. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/ round-2-submissions.

[7] A. Banerjee, C. Peikert, and A. Rosen, "Pseudorandom Functions and Lattices," *In EUROCRYPT 2012*, pp. 719-737, 2012.

[8] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM*, vol. 56, no. 6, 34, 2009.

[9] V. Lyubashevsky et al., "On ideal lattices and learning with errors over rings," *Int. Conf. Theory & Appl. of Crypto. Tech.*, pp. 1-23, 2010.

[10] J. Xie et al., "Special Session: The recent advance of hardware implementation of post-quantum cryptography," *IEEE VTS*, pp. 1-10, 2020.

[11] T. Pöppelmann et al., "Area optimization of lightweight lattice-based encryption on reconfigurable hardware," *ISCAS*, 2014, pp. 2796-2799.

[12] S.S. Roy et al., "Compact Ring-LWE cryptoprocessor," *CHES*, pp. 371-391, 2014.

[13] W. Liu et al., "Optimized schoolbook polynomial multiplication for compact lattice-based cryptography on FPGA," *IEEE TVLSI Syst.*, vol. 27, no. 10, pp. 2459-2463, 2019.

[14] A. Aysu et al., "Binary Ring-LWE hardware with power side-channel countermeasures," *DATE*, pp. 1253-1258, 2018.

[15] A. Karmakar et al., "Saber on ARM CCA-secure module lattice-based key encapsulation on ARM," *IACR Cryptology ePrint*, 2018:682, 2018.

[16] J. Mera et al., "Time-memory trade-off in Toom-Cook multiplication: an Application to Module-lattice based cryptography," *IACR TCHES*, vol. 2020, no. 2, pp. 222-244, 2020.

[17] J. Mera et al., "Compact domain-specific co-processor for accelerating module lattice-based KEM," *DAC*, pp. 1-6, 2020.

[18] V. Dang et al., "Implementing and benchmarking three lattice-based post-quantum cryptography algorithms using software/hardware codesign," *FPT 2019*, pp. 206-214, 2019.

[19] T. Fritzmann et al., "RISQ-V: Tightly coupled RISC-V accelerators for post-quantum cryptography," *Cryptology ePrint*, Report 2020/446.

[20] S. Roy and A. Basso, "High-speed instruction-set coprocessor for lattice-based key encapsulation mechanism: Saber in hardware," *IACR TCHES*, vol. 2020, no. 4, pp. 443-466, 2020.

[21] Y. Zhu et al., "LWRpro: An energy-efficient configurable825825crypto-processor for Module-LWR," *IEEE Trans. Circuits and Systems I: Regular Papers*, vol. 68, no. 3, pp. 1146–1159, 2021.

[22] A. Basso and S. Sinha Roy, "Optimized polynomial multiplier architectures for post-quantum KEM Saber," *DAC'21*, pp. 1-6, 2021.

[23] P. He, C.-Y. Lee, and J. Xie, "Compact coprocessor for KEM Saber: Novel scalable matrix originated processing," *The NIST Third Standardization Conference*, pages 1–16, 2021.

[24] V. B. Dang, K. Mohajerani, and K. Gaj, "High-speed hardware767767architectures and fair FPGA benchmarking of CRYSTALS-Kyber, NTRU, and Saber," *The NIST Third Standardization Conference*, pages 1–48, 2021

[25] D. Hofheinz et al., "A modular analysis of the Fujisaki-Okamoto transformation," *15th International Conference Theory of Cryptography Proceedings Part I*, vol. 10677, pp. 341-371. 2017.

[26] J. Pollard, "The fast Fourier transform in a finite field," *Mathematics of computation*, vol. 25, no. 114, pp. 365-374, 1971.

[27] J. L. Imaña, "LFSR-based bit-serial $GF(2^m)$ multipliers using irreducible trinomials" *IEEE Trans. Computers*, 2020 (early access).

[28] J. Xie et al., "Efficient hardware implementation of finite field arithmetic $AB + C$ for binary Ring-LWE based post-quantum cryptography," *IEEE Trans. Emerging Topics In Computing*, pp. 1-6, 2021. (accepted)

[29] T. Schneider et al., "Part I Towards combined hardware countermeasures against side-channel and fault-injection attacks," *Proc. Annu. Cryptol. Conf.*, pp. 302-332, 2016.